Ontological Generation of Filter Rules for Context Exchange in Autonomic Multimedia Networks

Steven Latré*, Sven van der Meer†, Filip De Turck*, John Strassner‡ and James Won-Ki Hong‡
*Ghent University - IBBT - Department of Information Technology, e-mail: steven.latre@intec.ugent.be
†Waterford Institute of Technology, TSSG, Waterford, Ireland
‡Pohang University of Science and Technology (POSTECH), Korea

***Abstract*—Network management has suffered from increases in business, system, and operational complexity. This has been exacerbated by the heterogeneity in management data as well as the high quality requirements of multimedia services. Autonomic networking manages this growing complexity by adding intelligence inside network nodes and network management applications. While most autonomic applications simply use a control loop to monitor and configure entities, our work is aimed at building a self-governing network that is able to fulfill the requirements of current and future services. This means that management applications need a detailed and dynamic view of the contextual status of the network nodes as a whole in order to adapt their behaviour to changing context. In this paper, we propose an algorithm to semi-automatically generate filter rules based on existing information in a network management information model. These filter rules are used to determine the set of contextual data that needs to be exchanged with other nodes. The algorithm exploits the reasoning capabilities of ontologies and relies on the introduction of additional semantic relationships to achieve a fine-grained context exchange model. Large scale evaluations were conducted to characterise the performance of this ontological approach.**

I. Introduction

In the last decade, networks have evolved from simple data packet forwarding to platforms that support complex multimedia services such as Network-Based Personal Video Recording and Broadcast TV. Each of these services has significant quality demands: they are very sensitive to packet loss and jitter, and require a substantial amount of bandwidth. As the quality perceived by the end user gives the most accurate view on the streamed service quality, operators are more and more focusing on this type of metric, commonly described as Quality of Experience (QoE).

The introduction of multimedia services together with the inherent heterogeneity in management data and programming models of today's network devices has led to increasing business and system complexity. By introducing more intelligence into the network and applications that manage network devices, a self-governing network can be realised. Figure 1 shows a simplified version of the FOCALE [1] autonomic architecture, which is made up of a set of distributable components connected by an enterprise service bus (ESB) that supports simple as well as semantic queries. An ESB is an event-driven message broker. The FOCALE implementation is a distributed *content*-based message and retrieval broker, meaning that it can take actions on the message and its content. The FOCALE

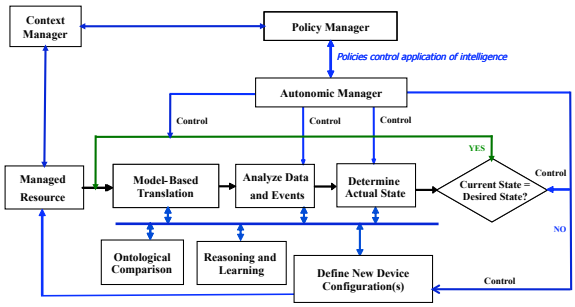

Fig. 1: Simplified FOCALE Autonomic Architecture.

Autonomic Manager uses this bus to orchestrate behaviour. It can support different types of knowledge acquisition and distribution (e.g., push, pull, and scheduled) and performs common processing before content is delivered to components. This enables components to register interest in knowledge in a more precise fashion, and thus reduce messaging overhead.

In FOCALE, the actual state of an entity is continuously compared with the desired state of that entity. If the comparison is equal, then monitoring continues; otherwise, one or more new control loops are formed to reconfigure the affected entity. The key to the FOCALE *adaptive control loops* is the interaction between the context manager, policy manager, and autonomic manager. Conceptually, the context manager detects changes in the network, or in user needs, or even in the business interactions; these context changes in turn activate an associated set of policies that define the functionality the autonomic manager should govern. This reconfigures one or more devices, so that the services and resources provided by the autonomic system can adapt to these new needs.

The unique context-aware control loops of FOCALE can use knowledge to affect local behaviour of an individual managed entity, or global behaviour of a set of managed entities. This depends on the distributed nature of knowledge. Based on its own behaviour, each entity has an associated set of contextual data. To organise the exchange of contextual information between entities, event notification services are used that enable each entity to expresses its interest in contextual information through filter rules. While this provides a clear architecture to organise the context exchange, the manual construction of these filter rules remains a challenging task.

This paper focuses on automating the generation of filter rules based on information models, in order to deliver the

appropriate contextual data to each entity [2], [3], [4]. The contributions of this paper are threefold. In order to automate the generation of the filter rules, we introduce extensions to the DEN-ng information model that model how distributed entities consume remote contextual data to optimise the QoE of services. Second, we define an algorithm to semi-automatically generate the required filter rules from information contained in the DEN-ng information model. The algorithm derives a baseline ontology from the DEN-ng information model, and defines semantic relationships that achieve a higher expressiveness. Third, we characterise the performance of the proposed solution by evaluating it on a large testbed that simulates an access network of more than 10,000 connected home gateways.

II. Related work

In recent years, several management information and data models have been proposed as a way to define a common networking lingua franca. These UML based models provide a way to share and reuse data as well as resolve interoperability problems among different technologies and vendor-specific implementations. Examples include the DMTF Common Information Model (CIM) [2], the TMForum's Information Framework (SID) [3] and the Autonomic Communication Forums (ACF) DEN-ng information model [4], which is a part of the FOCALE architecture. The CIM is in reality a *data model*, since it is not technology-neutral (e.g., it uses keys and weak relationships, which are database concepts) and is hard to extend (e.g., it does not use software patterns, which both the SID and DEN-ng do). The SID was partly based on DEN-ng v3.5; however, the directions of the TMForum changed, and DEN-ng was moved to the ACF. DEN-ng has a number of significant advantages over the CIM and SID; we refer to [5] for an overview of these.

The problem of organising content in a distributed environment has been the focus of research in Content Based Networks (CBNs) [6]. Work in [7] applies these CBNs to an autonomic communications environment to form a Knowledge Based Network (KBN). We believe the work presented in this paper complements this approach: while the KBN work focuses on semantic clustering of information [8] and extending current CBN solutions with semantic constructs [9], we focus on the automatic generation of filter rules through an ontological approach to provide information for KBNs.

We exploit the formal reasoning capabilities of ontologies to achieve a higher expressiveness and to automate the filter generation process. The value of using ontologies in network management has been argued in [10]; for example, ontologies have been used to provide interoperability between management domains [11]. In the field of autonomic communications, ontologies have been successfully applied to policy conflict analysis [12] and reasoning for autonomic networking [13].

III. Objective

In this section, we elaborate on the goal of the generation of filter rules through an exemplary scenario in QoE management. However, the same process can be used for organizing the exchange of contextual data between other distributed entities. To manage the QoE of multimedia services, networks often include QoE optimisers. We define a QoE optimiser as a component that alters the delivery of sessions to optimise a specific part of the QoE of that service. Typical examples are admission control mechanisms, SVC transcoders and retransmission mechanisms. In general, a number of QoE optimisers are deployed in the network and each QoE optimiser fulfills a particular function, and can both consume and produce contextual data. For example, an SVC transcoder requires information about the end-devices and network status to know which video quality level it needs to support. On the other hand, the decision taken by the SVC transcoder can be seen as a production of contextual data.

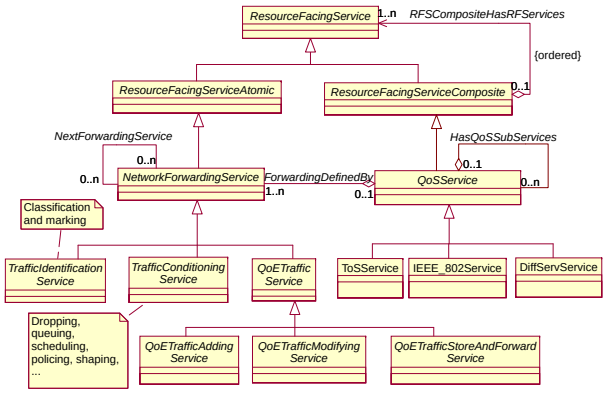

Fig. 2: DEN-ng modelling of QoE optimisers. The QoE Optimisers are modelled as a QoETrafficService, a new type of NetworkForwardingService

Contextual data generated by QoE optimisers can potentially be used by other QoE optimisers. For example, an admission control mechanism might be interested in the decisions taken by a SVC transcoder but will not be interested in which retransmissions a retransmission scheme performs. To extract the relevant contextual data out of the set of potentially interesting data, we use filter rules. In this scenario, these filter rules can be seen as queries that retrieve the output from a QoE optimiser. Hence, the filter rules express the contextual data that is of interest to other users. As the operation of QoE optimisers changes over time, so can the type of contextual data QoE optimisers they are interested in. In this article, we try to address this dynamicity in interests by automatically generating the filter rules.

IV. The DEN-ng QoE extension

This section describes the modeling of the QoE optimisation of multimedia services in DEN-ng.

A. Modelling QoE optimisers

DEN-ng separates the concept of a Service into *CustomerFacingServices* and *ResourceFacingServices*, which represent services that are visible to customers versus services that are internal to the network, respectively. For example, a VPN is typically visible to a customer, whereas the routing and

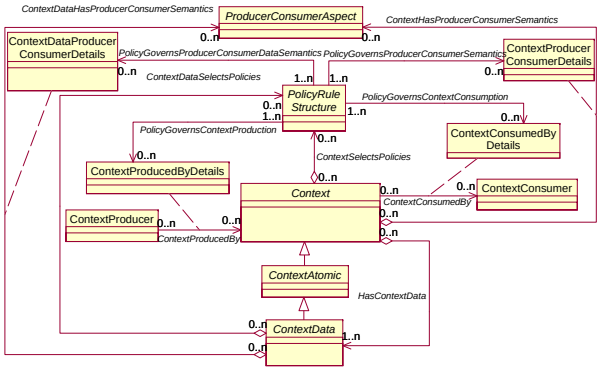

Fig. 3: DEN-ng modelling of context production and consumption.

forwarding protocols that the VPN uses are typically not visible to a customer.

Quality of Service (QoS) is modeled in DEN-ng as an architectural entity. There are currently three types of *NetworkForwardingServices*. A *TrafficIdentificationService* classifies and optionally marks a service; a *TrafficConditioningService* performs one or more operations on traffic, such as dropping, queuing, and scheduling; a *QoETrafficService*, which is a part of our extensions to DEN-ng, models the typical behaviour of QoE optimisers. The *QoETrafficService* distinguishes between QoE optimisers that introduce additional traffic to an existing session (i.e., *QoETrafficAddingService*), QoE optimisers that modify the actual payload of packets (i.e., *QoETrafficModifyingService*) and QoE optimisers that do nothing with the current session but can alter future sessions based on information about the current session (i.e., *QoETrafficStoreAndForwardService*). For example, a FEC encoder can be considered a *QoETrafficAddingService*, a video transcoder is a type of *QoETrafficModifyingService*, and a caching mechanism is an example of a *QoETrafficStoreAndForwardService*.

B. Context producer and consumer

The QoE optimisers modelled in the previous section can both generate and operate on contextual data. For example, a caching mechanism will require information about the popularity of the content it caches. At the same time, it will also generate contextual data about its decisions, such as which content it caches. The DEN-ng model represents the Context of an Entity by distinguishing between a collection of data, information, and knowledge that result from collecting measurements of and reasoning about that Entity. As illustrated in Figure 3, the *Context* and *ContextData* classes represent the concepts of whole (or assembled) context and partial (or component pieces of) context, respectively. This approach is unique in context modelling, and avoids the common weakness of representing context using an unorganised set of objects.

DEN-ng defines a *ContextProducer* and *ContextConsumer* to model the generation and consumption of contextual data. To link our new classes with the existing context model, two new associations have been added. The *ContextConsumedBy* and *ContextProducedBy* associations define the set of contextual data that is consumed and produced by a *ContextProducerService*, respectively. To govern the contextual data that is produced or consumed, DEN-ng defines policies through four associations (*PolicyGovernsContextProduction*, *PolicyGovernsContextConsumption*, *PolicyGovernsProducerConsumerDataSemantics* and *PolicyGovernsProducerConsumerSemantics*). These associations define the set of policy rules that are used to determine the contextual information that is produced or consumed as well as what *Context* or *ContextData* can influence products, services, resources and other managed entities that require context.

V. Orchestrating the context exchange

In this section, we propose a semi-automated process of generating filter rules that exploits the information already present in the DEN-ng information model and the formal reasoning capabilities of ontologies.

A. Motivating example for the use of ontologies

An UML based information model, such as DEN-ng, can represent most of the needed contextual data for making network management decisions. However, in some cases, additional semantic relationships are needed to characterise and reason on the available context. Therefore, we derive a baseline ontology from DEN-ng for reasoning. While there are cases in which traditional UML based information models are sufficient, we believe that the introduction of ontologies offers additional logic and reasoning power to make more efficient decisions. In particular, cases that require the ability to make and prove inferences are difficult to realise by using only UML, since it does not have any formal logic.

An example of this is tuning the amount of context that is requested from other nodes. Several QoE optimisers rely on a periodic update of information to make their decisions. Controlling the frequency and amount of content of these updates provide a trade-off: limit the communication overhead at the cost of losing accuracy. As another example, different services that co-exist may have different needs. An operator may choose to distinguish between these different services by increasing the update frequency of one service, and compensate for this by decreasing the update frequency of the other services. That way, the operator can fine tune the traffic conditioning of each service. These relations are difficult to model using UML, but can be easily represented by using a rule based ontological approach.

B. Automatic generation of filter rules

The proposed algorithm consists of a three step approach. In the first phase, potentially interesting context data is identified; these data are used to select an appropriate subset of the DEN-ng information model, which is then translated to an ontology. In the second phase, the information model is queried and relevant model elements are mapped to instances of the ontology. In the third phase, the ontology is queried and the filter rules are automatically generated based on the reasoning results. These steps are delineated in the following sections.

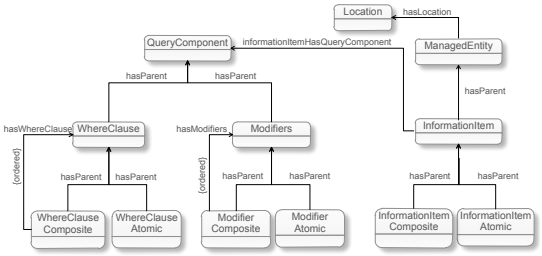

Fig. 4: The initial ontology used as the basis for converting the DEN-ng information model to an ontology based model for automatically generating the filter rules.

1) Phase I: Constructing the ontology T-BOX: In the first phase, the ontology T-BOX, which describes the properties and concepts in the ontology, is constructed. For this construction, a modified version of the algorithm described in [14] is used, which discusses a way to translate a UML model to an ontology. The modifications were made to accommodate the use case specific translation. A few steps were added, namely step 2 and 7-9:

Step 1 Manually tag the classes that have *potentially* relevant information in the DEN-ng information model. Typically, these classes will be subclasses of the *Context* or the *ContextData* classes in DEN-ng.

Step 2 The initial ontology, as depicted in Figure 4, is automatically constructed. This ontology defines a concept *InformationItem*, which is the root concept for the different pieces of context that can be queried. It is conceptually equivalent to the Context class in the DEN-ng information model. However, it can contain one or more *QueryComponents* that enable the queried result to be altered by typical query constructs.

Step 3 For every DEN-ng tagged class, an ontology concept is generated with the same name.

Step 4 All properties of the tagged classes are mapped to properties in the ontology.

Step 5 All associations of the tagged classes are mapped to properties in the ontology with restrictions to model the multiplicity of the association in UML.

Step 6 All subclasses in the information model are mapped to disjoint classes in the ontology.

Step 7 Optionally, extend the resulting ontology with semantic relationships that help determine which context data are needed. This is a task that should be done by a domain expert and makes up the manual phase in the process, together with the tagging of classes in DEN-ng (step 1).

Step 8 A concept called *InformationToQuery*, which has the *InformationItem* as parent but is not disjoint with the other subclasses is added to the ontology with the following SWRL rule as an initial definition: *InformationItem(?i)* $\wedge$ *hasLocation(?n)* $\wedge$ *isDifferent(?n,?l)* $\wedge$ *hasIP(?n,<IPAddress>)* $\wedge$ *hasBeenQueried(?i,FALSE)* $\rightarrow$ *InformationToBeQueried(?i)*. This basic rule defines all non-local information that have not been queried before as information that needs to be queried.

Step 9 If needed, more complex definitions of the *InformationToQuery* concept can be specified that build on the added semantic relationships and provide a more limited view of what needs to be queried. These definitions can be stated through rules and/or constraints, and can be realised using policies. An example of how these refinements can be defined is discussed in Section VI.

2) Phase II: Mapping DEN-ng instances to the ontologies A-BOX: Once the ontology T-BOX has been constructed, it is possible to translate the instance information present in the DEN-ng data model to the ontology A-BOX, describing the actual knowledge contained in the ontology. This is done via the following steps.

Note that the DEN-ng datamodel is a concrete implementation of the DEN-ng information model. Such a data model will employ a specific repository, programming language and protocol to provide access to the modelled entities. Only one DEN-ng information model is used; it serves as a template that models entities and their relationships. Different DEN-ng data models that reflect different implementation choices are derived from the DEN-ng information model. This ensures that different data models have a coherent and consistent definition of knowledge, even if they implement that knowledge differently.

Step 1 Information about all the nodes in the network is realised as instances of the *ManagedEntity* concept.

Step 2 The DEN-ng data model is queried for the instances of all tagged classes: the result is added as instances of the ontology.

Step 3 For each instance added in the previous step, the *hasLocation* property is established by querying the DEN-ng data model for the location of this item.

Step 4 Optionally, additional semantics to the existing context information can be manually introduced through query components.

Step 5 The DEN-ng data model is queried for applicable policies. For example, an applicable policy is a policy that has one or more of the added instances in the second step. In this example, a PolicySubject is a set of managed entities that represent the authority imposing policy. It can make policy decision and information requests, and it can direct policies to be enforced at a set of PolicyTargets. A PolicyTarget is a set of managed entities that a set of policies will be applied to. Since many different data models can be derived from the same information model, different data models can use the *same queries*, thereby enforcing consistency among heterogeneous data.

Step 6 Based on the definition of *InformationToQuery* an ontology reasoner is used to automatically classify which of the added ontology instances belong to the *InformationToQuery* concept.

3) Phase III: Automatic generation of filter rules: Once the instances belonging to the *InformationToQuery* concept have been automatically classified through reasoning, the filter rules can be generated as described in Algorithm 1. In the

algorithm, the getFilterRules function queries the constructed ontology and constructs a mapping between each node and the corresponding set of filter rules. For each node, the corresponding set of filter rules is constructed using the getFilterRulesByNode function. In this function, only those *InformationToQuery* individuals that have a *hasNode* property with the corresponding node are used. The filter rules are constructed by investigating the properties of the *InformationToQuery* item itself.

Algorithm 1 The algorithm for automatically generating the filter rules based on the constructed ontology.

FilterRule : $Name \times Modifiers \times WhereCondition$
FilterRuleMap : $Node \rightarrow \mathbb{P} FilterRule$
getFilterRules : $Ontology \rightarrow \mathbb{P} FilterRuleMap$

getFilterRules$(ont) \triangleq$
  $let\ \ nodes = getNodes(ont)$
  $let\ \ rulemap = \phi$
  $\forall n \in nodes:$
    $let\ \ rules = getFilterRulesByNode(ont, n)$
    $rulemap = rulemap \sqcup (n \rightarrow rules)$
  return $rulemap$

getFilerRulesByNode : $Ontology \times Node \rightarrow \mathbb{P} FilterRule$
getFilterRulesByNode $(ont, node) \triangleq$
  $let\ \ rules = \phi$
  $let\ \ items = getInformationToQuery(ont)$
  $\forall i \in items$
    if $hasLocation(i, node)$
    then
      $rules = rules \cup (name(i), mod(i), where(i))$
  return $rules$

VI. Access Network Scenario

In this section, we discuss how the process described in the previous section works for an access network scenario. We define four different types of QoE optimisers that are deployed on different nodes in the network: an admission control mechanism, a FEC mechanism, a retransmission mechanism, and an SVC transcoder. Our research is focused on current and future networks; hence, our architecture enables a reasoning component to be placed in either a network node or provided as part of a policy-based interaction. This allows the architect to choose how reasoning for supporting QoE management is realised on an application-specific basis. This reasoning component is responsible for detecting drops in the service delivery quality and taking the necessary corrective actions by deploying a set of QoE optimisers. QoE optimisers can take different approaches in fulfilling the same goal. For example, an admission control mechanism and an SVC transcoder can both be used to avoid congestion; however, the first does this by blocking sessions, while the latter decreases the bandwidth requirements of existing sessions. The reasoning component is responsible for selecting the appropriate QoE optimiser based on context and policy. The set of QoE optimisers changes according to the needs of changing context.

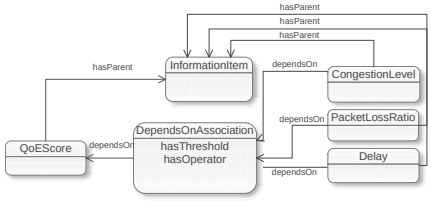

Fig. 5: Through the transitive $dependsOn$ property, we can model dependencies between context items.

A. Identifying relevant DEN-ng classes

In a first step, the relevant DEN-ng classes, are tagged and translated into an ontology's T-BOX and A-BOX. For this access network scenario, the classes to tag will contain the different QoE optimisers and the context they produce and consume (as a child class of DEN-ng's Context class). Once this translation is performed, we can then choose to add additional semantic relationships to improve the context communication and decision-making processes.

B. Introducing semantic relationships

Suppose we want to avoid requesting context data that is only needed when there is a QoE drop, because we want to minimise the communication overhead and a QoE drop occurs infrequently. We could request the QoE score, which is a summary of the performance of the service delivery. Based on this score, we can then retrieve additional information if required. This can be modelled by introducing a transitive object property in the ontology that states that one information item depends on another, as shown in Figure 5.

Ontology description 1 Restriction on the $InformationToQuery$ concept to incorporate the $dependsOn$ property

InformationItem $\wedge$ *dependsOn* **exactly** $0\ InformationItem$.

Ontology description 2 SWRL rule to make the inclusion of context information dependent on the value of another context item.

InformationItem(?child*)* $\wedge$ *InformationItemToQuery(*?root*)*
$\wedge$ *DependsOnAssociation(*?assoc*)* $\wedge$ *dependsOn(*?child,?root*)*
$\wedge$ *dependsOn(*?child,?assoc*)* $\wedge$ *hasThreshold(*?assoc,?th*)*
$\wedge$ *hasValue(*?root,?val*)* $\wedge$ *hasOperator(*?assoc,"LTE"*)*
$\wedge$ *lowerThanOrEqual(*?val,?th*)*
$\rightarrow$ *InformationItemToQuery(*?child*)*

Based on the $dependsOn$ property, we can then alter the definition of the $InformationItemToQuery$ concept to exclude the context items that depend on other context items as described in Ontology Description 1. Additionally, as described in Ontology Description 2, we can introduce a SWRL rule that includes the originally excluded context items.

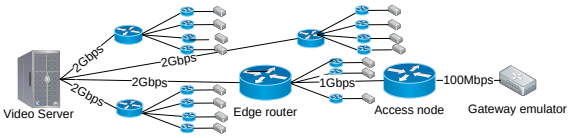

Fig. 6: Investigated topology consisting of 37 nodes which models an access network with 10,000+ subscribers.

C. Automatic classification of InformationToQuery

Once these additional semantic relationships have been defined, an ontology reasoner can be used to automatically classify which instances belong to the $InformationToQuery$ concept. In a first run, only one instance will be classified as an $InformationToQuery$ instance: the QoE score. Translating this into queries is done using Algorithm 1. If the value of the queried QoE score is below a threshold, additional context items (e.g., packet loss ratio, delay and congestion level), will be classified as $InformationToQuery$ instances in the second run as well.

VII. Performance evaluation

A. Test-bed implementation

We implemented a DEN-ng data model that models context in a network management environment. Our DEN-ng data model used the Hyper Structured Query Language Database to store data, and a Representational State Transfer based web-service to provide access to the data model. To support the automatic generation of filter rules, we used the Protege-OWL API library [15] as an ontology framework, and coupled it with the Pellet ontology reasoner [16] and the Jess rule engine [17].

We deployed the DEN-ng data model and context exchange architecture on a testbed of 37 nodes representing an access network with 10,000 connected home gateways. These home gateways are modelled as network edge nodes, where each node models the traffic of 625 home gateways. As shown in Figure 7, the investigated topology models an access network, where a server is streaming several video sessions to different home networks. We investigated both the network and application overhead. The first is expressed in terms of network delay and bandwidth, while the latter focuses on the time needed to reason over the ontology. All nodes in the topology were dual core AMD 2Ghz machines with 3GB RAM. During our experiments, we varied the number of services as well as the context requirements (i.e., the amount of requested contextual data) and the ontology structure.

B. Evaluation results

1) Influence on the network bandwidth and delay: We implemented the access network scenario as discussed in Section VI as an exemplary scenario with and without the use of the described additional semantic relationships. The configuration of these QoE optimisers can be determined for each service type independently (e.g., a VoD service and a Broadcast TV service). Alternatively, it is also possible to manage one particular service. For this test, we varied the number of managed service or service types and the number of nodes that take part in the context exchange process.

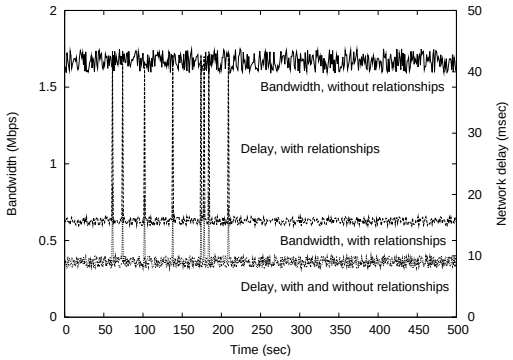

Fig. 7: Measured bandwidth and network delay over time for a context exchange process with and without semantic relationships.

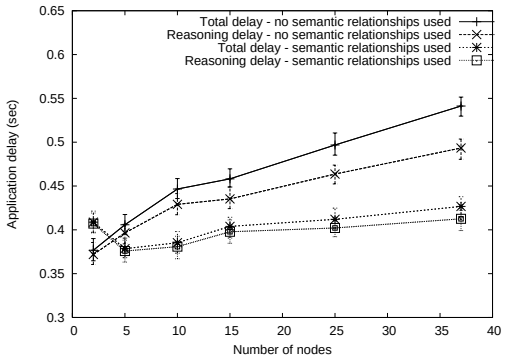

Fig. 8: Experienced overall and reasoning delay for a varying number of participating nodes and one managed service type.

Figure 7 illustrates both the measured network delay and bandwidth over time with and without additional semantic relationships. We define the network delay as being the delay needed to fetch one piece of contextual data from a remote node. If several queries are needed to obtain this data (e.g., because semantic relationships are also defined), the different queries are added together to define the delay value. In this test, 10 nodes were involved in the context exchange process, and 20 service types were managed. The context request frequency was 1 second. Without any additional semantic relationships, all the contextual data is requested each second which leads to an average bandwidth of 1.62 Mbps. The introduction of the $dependsOn$ property as described in Section VI dramatically reduces the bandwidth required. As only the QoE score is requested from other nodes the measured bandwidth is considerably lower (around 0.60 Mbps). Sometimes, additional contextual data needs to be requested (e.g., because a drop in the QoE score causes the SWRL rule to be triggered), resulting in a temporary increase in required bandwidth. For example in Figure 7, this happens around the 60 second time mark. As the SWRL rule is triggered, a second query is executed which introduces a short increase in network delay. Hence, the use of additional semantic relationships results in a decrease in the average bandwidth at the cost of occasional increases in network delay. This is the source of the spikes in Figure 7.

2) Influence of the context requirements on the reasoning delay: The delay introduced by the network is only a small

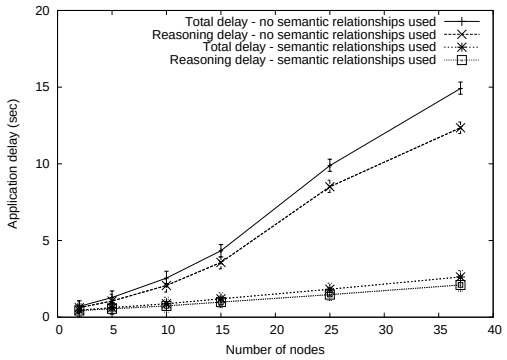

Fig. 9: Experienced overall and reasoning delay for a varying number of participating nodes and 50 managed service types.

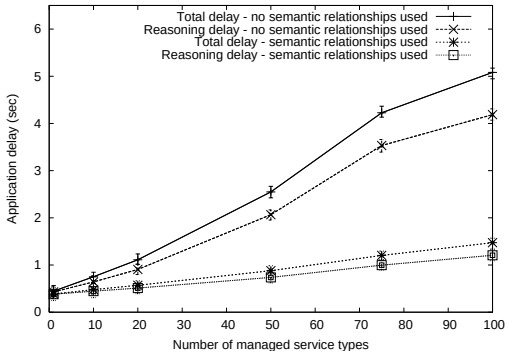

Fig. 10: Experienced overall and reasoning delay for a varying number of service types and 10 participating nodes.

fraction of the overall experienced delay; this is shown in Figure 8, which illustrates the overall delay and delay imposed by reasoning over the ontology for a varying number of nodes and 1 managed service type. As can be seen, the overall delay is primarily caused by the reasoning delay. An increase in the number of nodes leads to both an increase in reasoning delay and other types of delay (e.g. the network delay). This is expected since a larger number of nodes results in a larger ontology A-BOX, resulting in a more complex reasoning process. Additionally, a larger number of nodes will also result in a larger number of queries (as more nodes need to be contacted) and hence a larger network delay. The introduction of semantic relationships also has a positive effect on the reasoning delay. In an effort to decrease the communication overhead, the reasoning overhead is also decreased, because the use of reasoning results in fewer $InformationItem$ instances that are classified as relevant. For one managed service type and all 37 nodes participating in the exchange process, introducing semantic relationships results in a 20% decrease in overall delay (from 0.54 to 0.42 seconds).

The decrease in reasoning delay, experienced by introducing semantic relationships, is even more noticeable for a larger number of managed service types. This is depicted in Figure 9, which shows the overall delay and reasoning delay for a varying number of nodes and 50 managed service types. This figure shows two important trends. First, introducing semantic relationships reduces the average reasoning delay when the number of service types is increased. For 50 service types and 37 participating nodes, introducing semantic relationships results in a decrease of approximately 85% from 12 seconds to 1.81 seconds. This is because the effect of reducing the amount of information to analyse is multiplied. Second, a larger number of managed service types results in a larger overall and reasoning delay. Where the overall delay for 1 managed service type and 37 nodes was 0.54 seconds without any semantic relationships, the experienced delay for 50 managed service types and the same number of nodes is 14.86 seconds, an increase of 49%.

The effect of an increasing delay on an increasing number of managed service types is further illustrated in Figure 10, which shows the influence of a varying number of service types

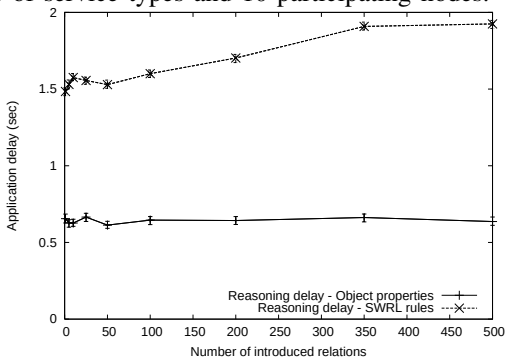

Fig. 11: Influence of the number of introduced semantic relationships on the reasoning delay

for 10 participating nodes. Increasing the number of service types complicates the reasoning process, as more context types will be present in the ontology. This more elaborate reasoning process is noticeable in an increase in reasoning delay. Similarly to Figure 8 and 9, the reasoning delay can be diminished by introducing additional semantic relationships to the ontology. For this test, a drop in reasoning delay of up to 71% can be achieved.

3) Influence of the ontology T-Box on the reasoning delay: All previous tests focused on the performance of the proposed solution for the access network model scenario (as discussed in Section VI). As such, only a limited amount of semantic relationships and SWRL rules were defined. Furthermore, the amount of contextual data being requested per managed service type and node was fixed. In the following test, we investigate the influence of the ontology T-Box on the performance of the reasoning delay by introducing more semantic relationships and sub-concepts of the *InformationItem* concept.

Figure 11 illustrates the influence of increasing the number of semantic relationships and sub-concepts on the reasoning delay. These semantic relationships can be introduced through SWRL rules or regular object properties. For this test, the number of participating nodes was fixed at 10, while the number of managed service types was set to 50. Furthermore, we assumed that every managed service type requests 10 context items per node. As can be seen, a larger number of introduced object properties does not have a significant influence on the reasoning delay.

Previous tests pointed out that introducing these object prop-

erties can greatly decrease the experienced reasoning delay. Here, we see that even a large number of object properties will not cause the reasoning delay to increase. Furthermore, the obtained standard deviation values are very low. There are two factors that contribute to this result. First, the use of caching performed by the Pellet reasoner greatly improves the performance. During deployment, modifications to the ontology are situated in the A-Box, which does not significantly complicate the reasoning process. Second, some of the introduced object properties are similar to the *dependsOn* property discussed in Section VI, meaning that their corresponding restriction on the *InformationToQuery* concept states that an *InformationItem* instance may not contain any of these properties. This means that the reasoning algorithm will stop classifying an instance as belonging to the *InformationQuery* concept when it finds such object properties.

When considering the introduced reasoning delay for a varying number of SWRL rules, we see that using these type of relations introduces an additional delay even for 1 SWRL rule. This delay is caused by calls made to a second type of reasoner, the Jess rule engine, because the first reasoner (Pellet) cannot apply SWRL rules onto the dataset. However, the delay experienced by increasing the amount of SWRL rules is considerably lower. The inclusion of 1 SWRL rule results in a delay of 0.92 seconds, while introduction of 500 additional SWRL rules introduces an additional delay of only 0.43 seconds on top of the 0.92 seconds.

In general, the overhead of the ontological approach is considerable: for example, for 25 participating nodes and 50 managed service types, the experienced reasoning delay with semantic relationships is above 2 seconds. This makes it hard to enable the use of ontologies in an on-line scenario where the context requirements need to be evaluated every second. However, while not immediately applicable to an on-line scenario, the ontological approach provides significant advantages to apply in an off-line scenario as it provides a way to automate the generation of filter rules where more complex semantic relationships can be achieved than with traditional UML based information models.

VIII. Conclusions & Future Work

In this paper, we defined a process to semi-automatically generate filter rules for the exchange of contextual data in autonomic networks. We modelled the context requirements of distributed entities in DEN-ng, which enables us to model the set of potentially relevant contextual data in a technology-neutral, object-oriented manner. To determine which contextual data from this set needs to be queried, an ontology is used that allows expressing semantic relationships that further refine the filter rule generation process. The proposed solution was deployed on a large scale testbed to evaluate the influence of the use of ontologies on the performance of the context exchange for a detailed multimedia access network scenario. It is shown that the introduction of additional semantic relationships results in both a reduction of network resources (i.e. bandwidth and delay) and application delay.

In future work, we are targeting to further apply additional machine learning techniques to automatically identify potential classes for consideration. Furthermore, we plan to introduce additional contextual elements to the baseline ontology. This will increase the expressiveness in the stated semantic relationships.

Acknowledgment

Steven Latré is funded by Ph.D grant of the Fund for Scientific Research, Flanders (FWO-V). This work is sponsored in part by the WCU (World Class University) program through the Korea Science and Engineering Foundation funded by the Ministry of Education.

References

[1] J. Strassner, N. Agoulmine, and E. Lehtihet, "Focale: A novel autonomic networking architecture," *International Transactions on Systems, Science, and Applications (ITSSA) Journal*, pp. 64–79, 2007.

[2] Distributed Management Task Force, "Common information model schema version 2.22," 2009. [Online]. Available: http://www.dmtf.org/standards/cim/

[3] TMForum, "Information framework (SID)," 2009. [Online]. Available: http://www.tmforum.org

[4] J. Strassner, *Policy-Based Network Management: Solutions for the Next Generation (The Morgan Kaufmann Series in Networking)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[5] J. Strassner, "Den-ng model overview," in *Joint ACF, EMANICS, and AutoI Workshop on Autonomic Management in the Future Internet*, 2008.

[6] R. Baldoni, R. Beraldi, L. Querzoni, and A. Virgilitto, "Efficient Publish/Subscribe through a Self-Organizing Broker Overlay and its Application to SIENA," in *The Computer Journal*, 2007.

[7] D. Lewis, D. O'Sullivan, K. Feeney, J. Keeney, and R. Power, "Ontology-based engineering for self-managing communications," in *Workshop on Modelling Autonomic Communications Environments*, 2006.

[8] J. Keeney, D. Jones, D. Roblek, D. Lewis, and D. O'Sullivan, "Knowledge-based semantic clustering," in *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*. New York, NY, USA: ACM, 2008, pp. 460–467.

[9] J. Keeney, D. Roblek, D. Jones, D. Lewis, and D. O'Sullivan, "Extending siena to support more expressive and flexible subscriptions," in *DEBS '08: Proceedings of the second international conference on Distributed event-based systems*. New York, NY, USA: ACM, 2008, pp. 35–46.

[10] J. E. L. de Vergara, V. A. Villagra, and J. Berrocal, "Applying the web ontology language to management information definitions," *Communications Magazine, IEEE*, vol. 42, no. 7, pp. 68–74, 2004.

[11] A. K. Y. Wong, P. Ray, N. Parameswaran, and J. Strassner, "Ontology mapping for the interoperability problem in network management," *Selected Areas in Communications, IEEE Journal on*, vol. 23, no. 10, pp. 2058–2068, 2005.

[12] S. Davy, B. Jennings, and J. Strassner, "Using an information model and associated ontology for selection of policies for conflict analysis," in *Proceedings of the Ninth IEEE International Workshop on Policies for Distributed Systems and Networks, POLICY*, 2008.

[13] J. Strassner, J. N. Souza, S. van der Meer, S. Davy, K. Barrett, D. Raymer, and S. Samudrala, "The design of a new policy model to support ontology-driven reasoning for autonomic networking," *J. Netw. Syst. Manage.*, vol. 17, no. 1-2, pp. 5–32, 2009.

[14] K. Barrett, S. Davy, J. Strassner, B. Jennings, S. van der Meer, and W. Donnelly, "A model based approach for policy tool generation and policy analysis," in *Global Information Infrastructure Symposium, 2007. GIIS 2007. First International*, 2007, pp. 99–105.

[15] "Protege-owl api." [Online]. Available: http://protege.stanford.edu/plugins/owl/api/

[16] E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical owl-dl reasoner," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, no. 2, pp. 51–53, June 2007.

[17] "Jess, the rule engine for the java platform." [Online]. Available: http://www.jessrules.com/